ThermalScope: A Practical Interrupt Side Channel Attack Based On Thermal Event Interrupts

Xin Zhang[1,2†], Zhi Zhang[3†], Qingni Shen[1,2*], Wenhao Wang[4], Yansong Gao[5], Zhuoxi Yang[1,2], Zhonghai Wu[1]
[1]School of Software and Microelectronics, Peking University,
[2]PKU-OCTA Laboratory for Blockchain and Privacy Computing, Peking University,
[3]The University of Western Australia, [4]Institute of Information Engineering, CAS, [5]Data61, CSIRO
Email: zhangxin00@stu.pku.edu.cn, zhi.zhang@uwa.edu.au, qingnishen@ss.pku.edu.cn, wangwenhao@iie.ac.cn, gao.yansong@hotmail.com, yangzx@stu.pku.edu.cn, wuzh@pku.edu.cn

ABSTRACT

While interrupts play a critical role in modern OSes, they have been exploited as a wide range of side channel attacks to break system confidentiality, such as keystroke interrupts, graphic interrupts and network interrupts. In this paper, we propose ThermalScope, a new side channel that exploits thermal event interrupts, which is adaptable for both native and browser scenarios and incorporates two heat amplifying techniques. The thermal event interrupts are activated only when the CPU package temperature reaches a fixed threshold that is determined by manufacturers. Our key observation is that workloads running on CPUs inevitably generates their distinct heat, which can be correlated with the thermal event interrupts. To demonstrate the viability of ThermalScope, we conduct a comprehensive evaluation on multiple Ubuntu OSes with different Intel-based CPUs. First, we show that the activation of thermal event interrupts correlates with the level of CPU temperature. We then apply ThermalScope to mount different side channel attacks, i.e., building covert channels with a transmission rate of 0.1 b/s, fingerprinting DNN model architectures with an accuracy of over 90% and breaking KASLR within 8.2 hours.

1 INTRODUCTION

Interrupts play a critical role as hardware resources in a modern operating system (OS), enabling OS process scheduler to preempt a running process and execute a corresponding interrupt handler. As some types of interrupts are triggered by users' activities, they have been abused as side channels.

Existing interrupt side channel attacks have exploited keystroke interrupts [6, 10, 11], graphic interrupts [7], and network interrupts [2, 13] to break system confidentiality. Specifically, keystroke interrupts will be generated when a user presses keyboard. As the interval between every two consecutive keystrokes correlates with user inputs, the inputs can be inferred via keystroke-based interrupts [6, 10, 11]. When GPU handles different workloads, it triggers a different number of graphic interrupts. With this observation, Ma et al. [7] exploit the graphic interrupts to identify different activities inside integrated/isolated GPUs. Last, network interrupts are triggered when websites are accessed. As different website access generates distinct network interrupts, they can be used to fingerprint websites [2, 13].

Our contributions: In this paper, we present ThermalScope, a new interrupt side channel attack through thermal event interrupts. This type of interrupts occurs when the temperature of an x86 CPU package is higher than a predetermined threshold, enabling the kernel to adjust throttling strategies in a timely manner. Our key observation is that workloads running on CPUs inevitably generates their distinct heat, which can be correlated with the thermal event interrupts. If the package temperature reaches the threshold when the victim code is executed, the thermal event interrupts will serve as a side channel to leak the victim's activity.

Observing Thermal Event Interrupts. To exploit the thermal event interrupts, we first need to observe them. We consider two real-world scenarios in this paper, i.e., *native* scenario and *browser* scenario. In the native scenario, aligned with [7, 13], we can use the `/proc/interrupts` interface to collect the statistics of thermal event interrupts. However, the interface can be disabled to mitigate aforementioned interrupt side channels [2]. As a bypass, we have identified `thermal_throttle`, an interface that allows us to obtain the number of thermal events that are logged by the thermal event interrupt handler. In the browser scenario where the attacker cannot access the system interfaces, we can use a loop-counting program to observe all incurred interrupts, including thermal event interrupts of our interest.

Triggering Thermal Event Interrupts. As the heat emanated by victim code may not be sufficient to cause the temperature to exceed the predetermined threshold, the second challenge is to reliably trigger the featured thermal event interrupts. To address this challenge, we propose two heat amplifying techniques: *Heat Padding* and *Heat Relaying*. First, on a multi-core system, the CPU package temperature is determined jointly by the workloads of all cores. With *Heat Padding*, the attacker executes a computing-intensive code concurrently with the victim code to push the temperature to reach the threshold. Second, as the decrease in temperature requires a significant amount of time, the temperature increments caused by a computing task can be observed even after it is stopped. With *Heat*

This work was supported by the National Natural Science Foundation of China (Grant No. 61672062). † Co-first authors. * Corresponding author.

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3656525

Relaying, the attacker executes another computing-intensive code right after a piece of attacker-controlled code is finished. When starting from a higher temperature, the attacker requires to execute fewer instructions, enabling she to distinguish the heat generated by the first piece of code.

To demonstrate that the thermal event interrupts are exploitable, we utilize ThermalScope to mount three side-channel attacks. First, we apply ThermalScope to build a covert channel with a transmission rate of 0.1 b/s. Second, we demonstrate that ThermalScope can be used to infer the activity of the victim process, fingerprinting the DNN model architectures with an average accuracy of over 90%. Last, ThermalScope successfully derandomizes kernel address-space layout randomization (KASLR) within 8.2 hours.

Summary of contributions: The contributions are summarized as follows:

- We propose ThermalScope, a new interrupt side channel through thermal event interrupts. To the best of our knowledge, we are the first to present the security implications of thermal event interrupts.
- ThermalScope is adaptable for both native and browser scenarios and incorporates two heat amplifying techniques. We have identified a new interface (i.e., `thermal_throttle`) to retrieve thermal event interrupts.
- We conduct three end-to-end attacks to evaluate ThermalScope with different Intel-based CPUs, including building covert channel, fingerprinting DNN model architectures, and breaking KASLR.

2 BACKGROUND AND RELATED WORK

2.1 Interrupt Side Channel Attack

As different interrupts are activated when the system handles different workloads, existing interrupt side channel attacks have exploited various types of interrupts, such as keystroke interrupts [6, 10, 11], graphic interrupts [7], and network interrupts [2, 13], to break system confidentiality.

First, when a victim presses the keyboard to input her secret, keystroke interrupts are activated to invoke the corresponding interrupt handler. Because the context needs to switch into kernel, the timestamp observed by users will become discontinuous. An unprivileged attacker can use the timing side channel to monitor the keystroke interrupts to infer the victim's input [6, 11]. Besides, when the corresponding interrupt handler code is loaded, the cache state changes. With this observation, KeyDrown [10] proposes to use cache side channel to detect the memory accesses to the interrupt handler code, thereby monitoring keystroke interrupts. To find targeted cache sets, they need the physical addresses of the keystroke interrupt handler.

Second, graphic interrupts are activated by a GPU when there is a need to handle specific tasks related to graphics processing, indicating significant events such as the completion of a graphics command or reporting a hardware error. The timing interval between every two graphic interrupts varies depending on the specific workloads being processed by the GPU. Ma et al. [7] exploit graphic interrupts as a separate side channel to leak the activities inside the integrated and isolated GPUs. They successfully mount several side-channel attacks under various scenarios, including fingerprinting documents, distinguishing applications, and recognizing non-GUI applications.

Last, the activation of network interrupts is related to network activities. For example, when data packets arrive at the network interface, network interrupts need to be activated to handle the reception and processing of the incoming data. Different websites can trigger different network activities, including distinctive network interrupts. An attacker can use these interrupts to fingerprint websites [2, 13], whose attack typically consists of two phases: an *offline preparation phase* and an *online classification phase*. In the *offline phase*, the attacker collects a number of interrupt traces and utilizes them to train the classifier. In the *online phase*, while the victim is opening a website, the attacker employs the pre-trained classifier to fingerprint which website the victim is visiting.

2.2 Thermal Event Interrupts

To implement a thermal throttling mechanism, x86 CPU cores within a package are equipped with a package thermal sensor (i.e., MSR 0x19c). When the sensor detects that the package temperature reaches a pre-determined threshold, thermal event interrupts will be triggered, generating context switches and enabling the OS kernel to cool off the package.

Specifically, when a workload is scheduled to execute, it will generates heat and the temperature of the CPU package on a whole will increase. If the thermal sensor detects that the temperature reaches the threshold, a thermal event interrupt will be generated and sent to every CPU core, switching their context to the kernel. As a response, the kernel logs the thermal event and takes actions against related cooling devices (e.g., increasing the fan speed, reducing the CPU frequency, etc). In Linux, the kernel has assigned an interrupt number to the thermal event interrupt and extended `/proc/interrupts` to record its statistics since version *2.6*.

3 THERMALSCOPE

In this section, we exploit thermal event interrupts to construct ThermalScope. First, we present the threat model and assumptions. Second, we introduce how we observe the thermal event interrupts in both native scenario and browser scenario. Last, we propose two heat amplifying techniques to trigger the thermal event interrupts.

3.1 Threat Model and Assumptions

In our threat model, we assume an unprivileged attacker. Thus, the attacker cannot make any modifications to a victim x86-based system. In the *native* scenario, the attacker can access local resources (e.g., system interfaces). In the *browser* scenario, the attacker is allowed to run their code in a sandbox.

To utilize our proposed heat amplifying techniques, the attacker needs to run her compute-intensive task to stress the CPU cores. Specifically, *Heat Padding* requires that the attacker runs her code concurrently with the victim code on another core and *Heat Relaying* requires that the attack code time-shares a core with the measured code.

To build covert channels, two user processes collude with each other. To fingerprint DNN model architectures, we assume that

the victim selects private models from PyTorch vision DNN architectures. To break KASLR, the victim system does not have any software bugs or vulnerabilities that enables the attacker to acquire a mapped kernel address.

3.2 Observing Thermal Event Interrupts

In this section, we discuss how to observe thermal event interrupts in native and browser scenarios.

In the native scenario, the `/proc/interrupts` interface provides statistics of various types of interrupts, which is available to unprivileged processes in Linux. Previous works have exploited the interface to observe targeted device interrupts, such as GPU [7]. Similar to them, we can also leverage this interface to retrieve the thermal event interrupts. To mitigate existing interrupt side channels, the interface can be disabled for unprivileged users. To bypass the mitigation, the `/sys/devices/system/cpu/cpuX/thermal_throttle` interface has been identified to provide information about our targeted thermal event interrupts. Specifically, this interface allows us to obtain the number of thermal events, whose logging depends on whether there is a thermal event interrupt occurring within a certain time interval.

In the browser scenario, the two interfaces above become inaccessible. To address this challenge, we use a loop-counting program to observe interrupts that have occurred, including thermal event interrupts. Specifically, the program loops to increase a counter value and sample its increment at fixed time intervals set by a timer. If the running core receives any interrupt (e.g., thermal event interrupts), context switch occurs, slowing down the counter's increment and thus signaling the occurrence of an interrupt. As high-resolution timers are crippled in browser scenario, the loop-counting program uses a low-resolution timer with a granularity of millisecond (e.g., 1 ms for Chrome and Firefox) to monitor interrupts.

3.3 Triggering Thermal Event Interrupts

In this section, we discuss two heat amplifying techniques to trigger thermal event interrupts.

Heat Padding: On a multi-core system, the CPU package temperature is determined jointly by the workloads of all cores. To amplify the temperature increments, the attacker can execute another compute-intensive code (called *padding code*) concurrently with the victim code. The padding code loops to execute the same instructions to generate constant heat, which is used to ensure that the package temperature exceeds the threshold to activate thermal event interrupts. Because the heat caused by the padding code is stable, the variation of thermal event interrupts can be attributed to the victim code. In this way, an attacker can infer victim activity by observing the induced thermal event interrupts.

As shown in Fig. 1, to construct *Heat Padding*, the attacker needs to run her padding code on a different core with the one occupied by the victim. By selecting an appropriate padding code, the package temperature is amplified near the threshold, causing distinctive thermal event interrupts during the execution of the victim program. She can infer the activity of the victim code via the activation of thermal event interrupts.

https://github.com/pytorch/vision

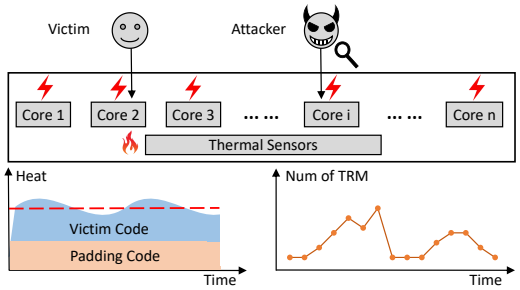


Figure 1: An illustration of using *Heat Padding* to exploit thermal event interrupts (TRM).

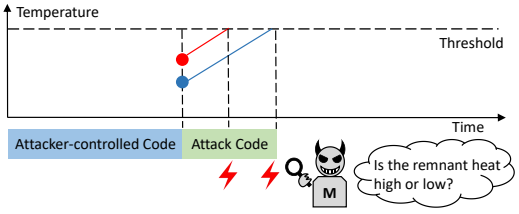


Figure 2: An illustration of using *Heat Relaying* to exploit thermal event interrupts.

Heat Relaying: Since the temperature increments caused by the computing task can be observed even after it is stopped, these increments can expose secret information to the following processes, especially when they share the same physical core [8]. This remnant heat allows the process to infer sensitive information from its predecessor, thereby violating time division. Unlikely other types of shared system resources that can be reset before context switching (e.g., CPU registers), as the decrease in temperature requires a significant amount of time, the remnant heat is hard to eliminate.

In this paper, we leverage the remnant heat to construct *Heat Relaying*, a new technique that obtains temperature observations by triggering featured thermal event interrupts. To achieve this, the attacker is required to execute her code on the same CPU core as the victim's code. As shown in Fig. 2, two pieces of code are executed sequentially. One piece is attacker-controlled code (e.g., a code snippet that accesses a kernel address in breaking KASLR) that executes a computational task related to user's secret. Obviously, according to the instructions and data executed, the attacker-controlled code will cause different heat (i.e., the red dot is higher than the blue dot). The other piece contains ThermalScope. After the execution of the attacker-controlled code, ThermalScope-based code loops to execute a specific operation until a thermal event interrupt is detected. Even if there is heat loss caused by the cooling device, the heat generated by the attacker code should be negatively correlated with the remnant heat. Considering that the heat generated by the attack code depends on the number of executed instructions, when starting from a higher temperature (the red dot), the attack code requires fewer instructions to execute. We use the number of executed instructions to guide our attack.

4 EVALUATION

In this section, we characterize ThermalScope and demonstrate how ThermalScope can be used to mount end-to-end attacks.

Machine	CPU	Kernel	OS
Xiaomi Air 13.3	Intel Core i5-8250U	5.15.0	Ubuntu 20.04.5
Lenovo Savior R9000	Intel Core i7-9750H	5.8.0	Ubuntu 22.04.1
Gigabyte z790 (motherboard)	Intel Core i7-14700K	5.15.0	Ubuntu 20.04.1

Table 1: System configurations.

Machine settings: We evaluate the performance of ThermalScope under various target systems as shown in Table 1, including different Intel-based CPUs. Unless otherwise stated, we use the default system configuration.

4.1 Characterizing ThermalScope

As outlined upon, based on observing a correlation between CPU package temperature and activation of thermal event interrupts, we construct ThermalScope. To show the effectiveness of our method in measuring temperature changes, we compare ThermalScope results with the temperature variation reported by the `hwmon` interface. `hwmon` is a mechanism for measuring and controlling the temperature of individual components of Linux-based machines. It provides a real-time measurement of the temperature for each CPU core, based on high resolution sampling of the integrated thermal sensors inside the CPU.

Experimental setup: In this experiment, `/proc/interrupts` is used to obtain the interrupt statstics and `thermal_throttle` is used to obtain the number of thermal events. Aligned with previous work [7], we choose a sampling period of 50 ms. To cause temperature increments, we use a certain number of *sqrt* operations to calculate the square root of a non-negative random number. We separately evaluate ThermalScope from two aspects on the Lenovo machine. First, we executes *sqrt* operations for 10 seconds and sleeps for 10 seconds in turns. In the same time, we use another process running on a separate core to record the temperature via `hwmon`, the number of thermal event interrupts, and the number of thermal events every 50 ms. In this way, we can directly observe the relationship between the activation of thermal event interrupts and CPU package temperature.

Second, we apply *Heat Relaying* technique to observe the relationship between the number of needed instructions and CPU starting temperatures. To achieve this, we have two pieces of code to execute. The first piece of code (i.e., attacker-controlled code) performs a random number of *sqrt* operations to cause different starting temperatures. Then, the second piece of code starts loops to execute *sqrt* operations and records the number of executed operations. In the same time, its child thread running on another core is used to monitor the activation of thermal event interrupts and notify the heating process to break their loops if a new interrupt is observed. We run the above steps for 1000 times.

Experimental results: As shown in Fig. 3, only when the temperature reaches a threshold (e.g., 80 °C in our Xiaomi machine and 90 °C in our Lenovo machine), the thermal event interrupts are activated and the thermal events are logged. Besides, the number of thermal events is less than thermal event interrupts, because the logging frequency of thermal events is limited by kernel. If there are two thermal events in a logging cycle, the kernel will only log them once. Fig. 4 shows the relationship between heating time and starting temperature. These two values approximately satisfy a linear relationship. The lower the starting temperature is, the more *sqrt* operations are executed. So the attacker can use the number of executed operations to infer the starting temperature that depends on the system activity.

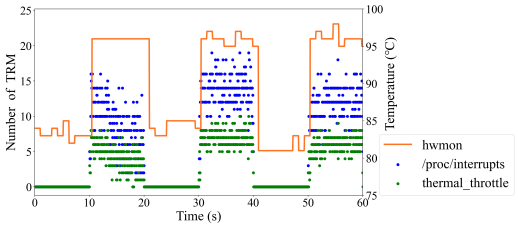


Figure 3: The relationship between the number of thermal event interrupts (TRM), the number of package thermal events, and temperature (through `hwmon`). Only when the CPU package temperature reaches a threshold, the thermal event interrupts are activated and the thermal events are logged.

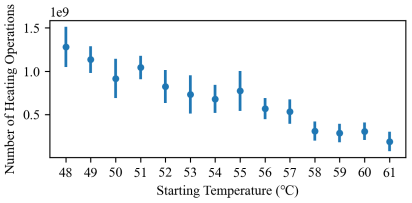


Figure 4: An approximate linear relationship between the heating time and starting temperature (through `hwmon`).

4.2 Building Covert Channels

A covert channel [1, 8] requires a pair of colluding sender and receiver, which are usually not allowed to communicate over normal channels. To implement a ThermalScope-based covert channel, a sender actively executes *sqrt* operations on random numbers to stress its CPU cores. Depending on the sender's intention to send bit '0' or bit '1', the child threads create different heats. The receiver uses the `thermal_throttle` interface to obtain the number of thermal event interrupts. To ensure that the thermal event interrupts are activated when sending bit '1', the sender applies *Heat Padding* to amplify the temperature increments.

Experimental setup: For each test, we transmit 1 kB of random data between two unprivileged processes running on different cores of the Lenovo machine. The sender controls the execution number of *sqrt* operations to control the temperature changes. To transmit a '1', the sender executes *sqrt* operations for 5 seconds and sleeps for 5 seconds. To transmit a '0', it sleeps for 10 seconds. Besides, the sender uses a process running on a separate core to continuously increment a counter, ensuring that the thermal event interrupts can be activated when sending bit '1'. The receiver obtains the increments of thermal event interrupts every ten seconds via `/proc/interrupts` or `thermal_throttle`. By comparing with a threshold, it identifies the increments as bit '0' or '1'.

Experimental results: Our PoC code achieves a transmission rate of 0.1 bit/s with a bit error rate of 0.2% for `/proc/interrupts` and 0.7% for `thermal_throttle` interface (average across 10 runs) in our Lenovo machine. According to our observation, the main source

of bit error is the remnant heat. When the sender continuously sends multiple '1' s, the temperature gradually increases and hampers timely cooling, thereby impeding the transmission of subsequent bits. However, this issue can be mitigated by extending the sleep time, albeit at the expense of reduced bandwidth.

4.3 Fingerprinting DNN Model Architectures

To design an effective DNN model, developers need to spend significant time and energy in searching and fine-tuning model architectures. The optimized model structures are always considered a confidential property [3]. Furthermore, a known DNN architecture can also help to mount adversarial transfer attacks [9], where the attacker requires a substitute model to generate her adversarial examples. The attack success rate depends on the architectural similarity between the victim model and the substitute model.

Our key insight of using ThermalScope to fingerprint DNN architecture is that there are a large number of matrix operations executed during the inference process of DNN models, which causes different pattern of heat. We can use ThermalScope as a side channel to fingerprint the DNN model structures. Aligned with previous work [3, 9, 12], our model fingerprinting attack has two phases: an *offline preparation* phase and an *online classification* phase. In the *offline preparation* phase, we collect the thermal event interrupt traces to build a series of well-trained classifiers, which can translate a trace to its corresponding model architecture. In the *online classification* phase, we query a black-box target model running on the victim machine to trigger its model inference and collect our side channel traces. Then we use the offline-trained classifiers to fingerprint the model architectures.

Experimental setup: We carry out our experiments in all the tested machines. The victim model uses PyTorch 1.13.0 with Python version 3.9.13 as its underlying deep learning framework. The attacker process and the victim process run on separate physical cores. Aligned with [9, 12], the victim models are selected from the PyTorch vision DNN architectures, including 36 architectures over 9 diverse architecture families. All the pretrained models expect the input images to be 224x224 and normalized in the same way. We run inference on the ImageNet ILSVRC Test set images. The victim runs a DNN model in series for 25 seconds. We pin the attack process and the victim process to separate cores to avoid scheduling contention.

Model Family	Model Architecture
VGG	VGG11, VGG13, VGG16, VGG19 VGG11_bn, VGG13_bn, VGG16_bn, VGG19_bn
ResNet	ResNet18, ResNet34, ResNet50, ResNet101, ResNet152 Wide_ResNet50_2, Wide_ResNet101_2, ResNext50_32x4d, ResNext101_32x8d
SqueezeNet	SqueezeNet1_0, SqueezeNet1_1
DenseNet	DenseNet121, DenseNet161, DenseNet 169, DenseNet201
ShuffleNet	ShuffleNet_v2_x0_5, ShuffleNet_v2_x1_0 ShuffleNet_v2_x1_5, ShuffleNet_v2_x2_0
MnasNet	MnasNet0_5, MnasNet0_75, MnasNet1_0, MnasNet1_3
MobileNet	MobileNet_v2, MobileNet_v3_large, MobileNet_v3_small
AlexNet	AlexNet
GoogleNet	GoogleNet

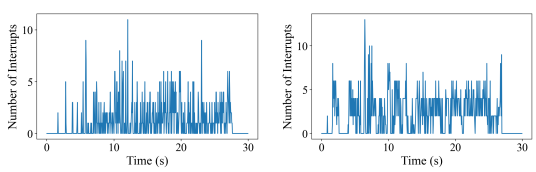
Table 2: The 36 PyTorch vision architectures which were used in evaluating our fingerprinting attack.

Aligned with [12], we run the inferences in series (not in a batch) because an attacker would not have control over the batch size input to the victim model.

Under the native scenario, we sample the increment of thermal event interrupts from `/proc/interrupts` by default at a fixed interval of 50 ms and collect 1,000 points for each attack. Under the browser scenario, a loop counting program is used to continuously increment a counter value. We sample it at a fixed interval of 10 ms, collecting 5,000 points for each attack. Because the loop-counting attack code measures instruction throughput, which can be impacted by processor frequency scaling, we use the Linux command `cpufreq-set` to fix the CPU frequency in this setting. Besides, to show the effectiveness of our method, we also test our method in other settings including observing the other interrupts, overall interrupts, and the thermal events. In the first two settings, We separately use `/proc/interrupts` to obtain the number of interrupts other than thermal event interrupts and the number of the overall interrupts on the attack core at a fixed interval of 50 ms. In the thermal events setting, we use the `thermal_throttle` interface to obtain the number of package thermal events on the attack core at a fixed interval of 50 ms. For each setting, we record 100 traces for each of the 36 model architectures.

We feed the collected side channel traces without any preprocessing into the classifiers, including Support Vector Machine (SVM), Logistic Regression (LR), Random Forest (RForest), and k-Nearest Neighbor (k-NN), to support our decision. For the SVM classifier, we choose a linear kernel function, and the soft margin constant is set to 10. For LR classifier, the range of penalty parameter C is 1, 10, 100, 1000. For the RForest classifier, we set the number of trees as 100 and the maximum depth as 32. For the k-NN classifier, we set the number of nearest neighbors as 10. For each classifier, we perform the 10-fold cross-validation during the evaluation, where nine folds are used as the training data, and the remaining one fold is retained as the testing data.

Experimental results: As shown in Fig. 5, each of the selected models exhibits a unique pattern, which depends on the unique heat caused by the matrix operations in the model inference phase. Table 3 shows the accuracy of various classifiers in fingerprinting the DNN model architectures. For the two best classifiers (i.e., RForest and SVM), the overall accuracy across the three tested machines is over 90% in the two scenarios. Table 4 shows the accuracy of our RForest classifier under different settings. Without thermal event interrupts, using the other interrupts can only achieve a low success rate of less than 50%. Besides, the average success rate of overall interrupts setting is 89.8%, which explains why our attack works in browser scenarios. The reason for the slightly higher success rate of attacks in browser scenarios is that the loop-counting program essentially observes the handling time of overall interrupts, rather than the number, thus obtaining more information. Last,



(a) ResNet50 (b) VGG19

Figure 5: Thermal event interrupt patterns caused by different DNN model inferences.

Scenario \ Classifier	Native Scenario				Browser Scenario			
	Xiaomi	Lenovo	Gigabyte	**Average**	Xiaomi	Lenovo	Gigabyte	**Average**
SVM	0.940	0.980	0.821	0.914	**0.937**	**0.929**	**0.930**	0.932
LR	0.929	0.976	0.788	0.898	0.923	0.918	0.898	0.913
RForest	**0.964**	**0.982**	**0.838**	0.928	0.933	0.923	0.921	0.926
k-NN	0.879	0.930	0.784	0.864	0.912	0.904	0.883	0.900

Table 3: Classification accuracy across 10-fold cross validation for DNN model fingerprinting.

Setting	Xiaomi	Lenovo	Gigabyte	**Average**
Thermal Event Interrupts	0.964	0.982	0.838	0.928
Other Interrupts	0.418	0.407	0.382	0.402
Overvall Interrupts	0.927	0.941	0.825	0.898
Thermal Events	0.955	0.984	0.833	0.924

Table 4: The impact of the type of interrupts (RForest is used as the classifier).

the success rate of using `thermal_throttle` interface to perform attack is over 90%, indicating that our attack still works even if the `/proc/interrupts` interface is disabled.

4.4 Breaking KASLR

In Linux, KASLR is implemented to randomize the base address of the text segment at every boot time, aligning it with a 2 MB boundary and mapping it to an address within a range of 1 GB. If an attacker wants to determine the text base address, a maximum of 512 times of guessing is needed. However, prior work [4, 5] has shown that a side channel leakage occurs when executing *prefetch* instructions on a memory address, depending on whether that address is mapped or not. Our key idea of breaking KASLR using ThermalScope is that the activation of thermal event interrupts also depends on the execution of *prefetch* instructions. We can use ThermalScope to distinguish the *prefetch* instructions to mapped address from the *prefetch* instructions to unmapped address.

Experimental setup: There are two configurable parameters in our prototype: the number of `prefetch` instructions to each possible address and the CPU core that is controlled by the attacker to amplify the heat. We carry out our experiments on the Lenovo machine listed in Table 1. We run our measurement 512 times, prefetching data from address in 2 MB intervals within the range 0xffffffff80000000 to 0xffffffffc0000000. For each possible address, the attacker-controlled code executes 3 billion *prefetch* instructions. Then, the attack code, which time-shares the same physical core with the attacker-controlled code, continuously computes *sqrt* functions on a random number until the thermal event interrupt is activated. The attacker uses the number of executed *sqrt* operations to distinguish whether the address is mapped or not. Before each try, we sleep for 30 seconds to cool down.

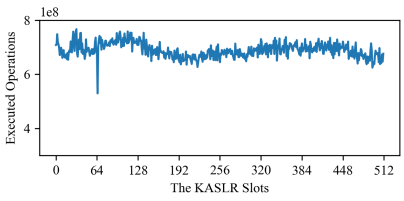


Figure 6: Number of executed *sqrt* operations when we apply heat relaying on every possible base address.

Experimental results: Fig. 6 presents the results of our experiment conducted on the Lenovo machine. The entire attack process takes 8.2 hours. When prefetching a mapped memory, the execution of *prefetch* instruction is faster and thus the starting temperature for *Heat Relaying* is higher than others. As a result, fewer *sqrt* operations are needed in *Heat Relaying* phase to make the package temperature reach the fixed threshold. Obviously, the slot 64 corresponds to the mapped address.

5 CONCLUSION

In this paper, we introduced a new side-channel attack, i.e., ThermalScope, which abuses thermal event interrupts as a side-channel. These thermal event interrupts are activated only when the CPU package temperature reaches a fixed threshold that is determined by CPU manufacturers. ThermalScope is adaptable for both native and browser scenarios and incorporates two heat amplifying techniques. To demonstrate the viability of ThermalScope, we evaluated ThermalScope on multiple Ubuntu operating systems with different Intel-based CPUs. The experimental results demonstrate that ThermalScope can be applied to establish covert channels, fingerprint DNN model architectures, and circumvent KASLR.

REFERENCES

[1] D. B. Bartolini, P. Miedl, and L. Thiele. On the capacity of thermal covert channels in multicores. In *European Conference on Computer Systems*, 2016.
[2] J. Cook, J. Drean, J. Behrens, and M. Yan. There's always a bigger fish: A clarifying analysis of a machine-learning-assisted side-channel attack. In *International Symposium on Computer Architecture*, page 204–217, 2022.
[3] Y. Gao, H. Qiu, Z. Zhang, B. Wang, H. Ma, A. Abuadbba, M. Xue, A. Fu, and S. Nepal. Deeptheft: Stealing dnn model architectures through power side channel. In *IEEE Symposium on Security and Privacy*, 2024.
[4] D. Gruss, C. Maurice, A. Fogh, M. Lipp, and S. Mangard. Prefetch side-channel attacks: Bypassing smap and kernel aslr. In *ACM SIGSAC Conference on Computer and Communications Security*, page 368–379, 2016.
[5] T. Kim and Y. Shin. Thermalbleed: A practical thermal side-channel attack. *IEEE Access*, 2022.
[6] M. Lipp, D. Gruss, M. Schwarz, D. Bidner, C. Maurice, and S. Mangard. Practical keystroke timing attacks in sandboxed javascript. In *European Symposium on Research in Computer Security*, pages 191–209, 2017.
[7] H. Ma, J. Tian, D. Gao, and C. Jia. On the effectiveness of using graphics interrupt as a side channel for user behavior snooping. *IEEE Transactions on Dependable and Secure Computing*, pages 3257–3270, 2021.
[8] R. J. Masti, D. Rai, A. Ranganathan, C. Müller, L. Thiele, and S. Capkun. Thermal covert channels on multi-core platforms. In *USENIX Security Symposium*, 2015.
[9] K. Patwari, S. M. Hafiz, H. Wang, H. Homayoun, Z. Shafiq, and C.-N. Chuah. Dnn model architecture fingerprinting attack on cpu-gpu edge devices. In *European Symposium on Security and Privacy (EuroS&P)*, pages 337–355, 2022.
[10] M. Schwarz, M. Lipp, D. Gruss, S. Weiser, C. Maurice, R. Spreitzer, and S. Mangard. Keydrown: Eliminating software-based keystroke timing side-channel attacks. In *Network and Distributed System Security Symposium*, 2018.
[11] J. Trostle. Timing attacks against trusted path. In *IEEE Symposium on Security and Privacy*, pages 125–134, 1998.
[12] J. O. Weiss, T. Alves, and S. Kundu. Ezclone: Improving dnn model extraction attack via shape distillation from gpu execution profiles. *arXiv preprint*, 2023.
[13] R. Zhang, T. Kim, D. Weber, and M. Schwarz. (M)WAIT for It: Bridging the Gap between Microarchitectural and Architectural Side Channels. In *USENIX Security Symposium*, 2023.